Formal verification of safety protocol in train control system

ZHANG Yan[1], TANG Tao[1*], LI KePing[1], MERA Jose Manuel[2], ZHU Li[1], ZHAO Lin[1] & XU TianHua[1]

[1] *State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China;*
[2] *Railway Technologies Research Centre, Universidad Politécnica de Madrid, Madrid 228006, Spain*

Received September 6, 2010; accepted August 8, 2011; published online September 30, 2011

In order to satisfy the safety-critical requirements, the train control system (TCS) often employs a layered safety communication protocol to provide reliable services. However, both description and verification of the safety protocols may be formidable due to the system complexity. In this paper, interface automata (IA) are used to describe the safety service interface behaviors of safety communication protocol. A formal verification method is proposed to describe the safety communication protocols using IA and translate IA model into PROMELA model so that the protocols can be verified by the model checker SPIN. A case study of using this method to describe and verify a safety communication protocol is included. The verification results illustrate that the proposed method is effective to describe the safety protocols and verify deadlocks, livelocks and several mandatory consistency properties. A prototype of safety protocols is also developed based on the presented formally verifying method.

train control system, safety communication protocol, interface automata, verification

Citation: Zhang Y, Tang T, Li K P, et al. Formal verification of safety protocol in train control system. Sci China Tech Sci, 2011, 54: 3078–3090, doi: 10.1007/s11431-011-4562-2

1 Introduction

Train control system (TCS) supporting safety-critical applications [1–4] often employs a layered protocol to provide various safety services [5–7]. A safety layer is required to be inserted between a transport layer and an application layer. The service interface between the application layer and the safety layer gives the data flows to/from the safety layer, which provides safety services. Since such a stack of the safety protocols may form the basis for numerous critical applications [8], their verification for safety concerns warrants strong guarantees of correctness and rigorous formal verification methods [9].

In the past decades, a number of efforts have been made in describing and verifying the protocols using formal method. Lee et al. [10–12] described a TCS protocol using label transition system (LTS) and verified it by model checking. Katsaros [13] used colored petri nets (CPN) and the CPN Tools environment to edit an electronic payment protocol model and verified the CPN model by computation tree logic (CTL) based model checking. Sinha et al. [8, 14, 15] showed how a complex protocol could be easily formalized by the reuse of the formal specification and verification of individual functional primitives of the chosen protocol. However, our observation is that these methods cannot describe the interface behaviors between adjacent layers explicitly, which makes them not very suitable for modeling and verifying the safety service interface behaviors of the safety protocols.

Ouzzif [16] presented a formal description of a floor control protocol using finite state machine (FSM) and

*Corresponding author (email: ttang@bjtu.edu.cn)

checked it by the SPIN verification tool. He depicted the message sequence chart (MSC) as a script of interactions among the components. Schafer et al. [17] and Inverardi et al. [18] described the component behaviors characteristics using unified modeling language (UML) state diagram and translated them into some PROMELA models. Meanwhile, the models of the UML interactive and sequential diagrams are used to describe the dynamic interactive relationships between the components. Finally, they were described by some linear temporal logic (LTL) formulas and verified by SPIN. But a main safety property of safety protocols in TCS is the absence of deadlocks. These methods did not mention how to analyze the deadlocks in the system specification using the counterexample given by SPIN. Moreover, UML state diagram is a semi-formal means of description without a mathematical basis, which might result in ambiguity and unsafety [8].

In order to satisfy the specific requirements of the safety communication protocols, interface automata (IA) are chosen precisely because of their characteristics. IA are good choices to describe the layered safety protocols due to the following features. (1) They are good at describing the safety service interface behaviors of the layered protocols. (2) IA are formal description methods with a mathematical basis. (3) They allow the existence of the blocking states, which can be used to detect and analyze the deadlocks. As a modeling language of SPIN, PROMELA can describe all the elements of IA. The translation rules from the IA model to the PROMELA model can also be defined to guarantee the consistent performance. The developed method relies on the applications of IA and SPIN for (1) abstracting and describing the layered safety protocols with a layer-based formal method, (2) detecting deadlocks and livelocks in the safety protocol behavior specifications, and (3) verifying consistency properties of the safety protocols by the LTL based model checking. The advantage of this method is that it can describe the safety service interface behaviors explicitly. It is also very expert at analyzing the causes of the deadlocks in the specification of the safety protocols in TCS. To illustrate the presented method, a TCS safety protocol is chosen as a case study. This protocol highlights the layer characteristics of safety protocols that typically provide the reliable safety services. In addition, an implementation of the safety communication protocol is described.

2 Interface automata

IA were proposed by Alfaro [19]. They are formal tools for modeling and analyzing the high level of specifications [20–24]. Unlike input/output (I/O) automata [25], IA have assumption environment. IA do not require any input-enabling, and allow the existence of blocking states, which are suitable for modeling the terminating processes and checking whether the assumption environment is right [19]. Chakrabarti et al. [22, 23] used extended IA to describe the more complicated interface behaviors. Edward [24] proposed some extensions to the theory of IA to define the interaction types and the dynamic behaviors of the components. An interface automaton P is defined as follows:

Definition 1. An interface automaton $P=(V_P, V_P^{\text{Init}}, A_P^I, A_P^O, A_P^H, \Gamma_P)$ consists of the following elements:

- V_P is a set of states; each state is symbolized as $v_i (0 \leqslant i \leqslant |V_P|)$.
- $V_P^{\text{Init}} \subseteq V_P$ is a set of initial states; one V_P^{Init} contains at most one state; if $V_P^{Init} = \phi$, then P is called empty.
- A_P^I, A_P^O, A_P^H are mutually disjoint sets of input, output and internal actions; $A_P = A_P{}^I \cup A_P{}^O \cup A_P{}^H$ is the set of all actions; if $a \in A_P{}^I$ (resp. $a \in A_P{}^O$, $a \in A_P{}^H$), then (v,a,v') is called an input (resp. output, internal) step.
- $\Gamma_P \subseteq V_P \times A_P \times V_P$ is a set of steps.

The interface automation P is closed if it has only internal actions, otherwise P is open. An action $a \in A_P$ is enabled at the state $v \in V_P$, if there is a step $(v,a,v') \in \Gamma_P$ for some $v' \in V_P$. It is indicated that the subsets of input, output, and internal actions are enabled at the state v with $A_P^I(v)$, $A_P^O(v)$, $A_P^H(v)$ respectively, and $A_P(v) = A_P^I(v) \cup A_P^O(v) \cup A_P^H(v)$. Unlike the I/O automata, an interface automaton is not required to be input enabled (it is not required that $A_P^I(v) = A_P^I$ for all states of v). The set $A_P^I(v)$ of the enabled input actions appoints which inputs are accepted at the state v. The inputs in $A_P^I \backslash A_P^I(v)$ are called the illegal inputs at the state v.

Definition 2. Two interface automata P and Q are composable if

$$\begin{aligned} &A_P^I \cap A_Q^I = \phi, \quad A_P^H \cap A_Q = \phi, \\ &A_Q^H \cap A_P = \phi, \quad A_P^O \cap A_Q^O = \phi. \end{aligned} \tag{1}$$

It makes

$$\text{shared}(P,Q) = A_P \cap A_Q. \tag{2}$$

Note that if two interface automata P and Q are composable, $\text{shared}(P,Q) = (A_P^I \cap A_Q^O) \cup (A_P^O \cap A_Q^I)$ and their product $P \otimes Q$ as defined by Alfaro et al. [19] is also an interface automaton.

Definition 3. Given two composable IA P and Q, the set $\text{Illegal}(P,Q) \subseteq V_P \times V_Q$ of the illegal states of $P \otimes Q$ is defined by

$$\text{Illegal}(P,Q)=\left\{(v,u)\in V_P\times V_Q \mid \exists a\in \text{shared}(P,Q)\times\left(\begin{array}{c} a\in A_P^O(v)\wedge a\notin A_Q^I(u) \\ \vee \\ a\in A_Q^O(u)\wedge a\notin A_P^I(v)\end{array}\right)\right\}. \quad (3)$$

Since IA are not necessarily input-enabled, in the product of two IA P and Q ($P\otimes Q$), one of the automata may produce an output action that is the input action of the other automaton, but it is not accepted. The set Illegal(P, Q) of the states of $P\otimes Q$ is called the illegal states of the product. Alfaro et al. [19] proved that some illegal states could be reachable in some certain environment. If there is no arc from the reachable illegal state to other states, the system is caused to terminate. IA have the illegal state and allow the environment assumption, which can be used to describe and analyze the implicit deadlocks in the specification of the safety protocols.

3 A method to verify the safety protocols

3.1 Method workflow

The workflow, a common waterfall model to verify the safety protocols, is shown in Figure 1. IA, UML sequence diagram and SPIN are combined and applied to do most of the work. In the first step, some critical scenario-based safety service interactions of the safety protocols are described by UML sequence diagram. Then some consistency verifying problems can be chosen from these UML sequence diagram models. IA are restricted and used to describe safety service interface behaviors of the components by abstracting away the details of computation referring to the chosen consistency verifying problems. Model Translation involves the translation from IA model to PROMELA model and the description of the consistency verifying problems using LTL. The description of the verifying problems must refer to the PROMELA model. Deadlocks and livelocks are the properties in a system, which cannot be described and checked using LTL. In addition, the checking of them is crucial for correctly expressing the LTL-based formulae, which is used to verify the protocol properties. Therefore the deadlocks and livelocks checking should be performed before the LTL properties verification. Based on the deadlocks and livelocks analysis, the LTL properties are able to be verified using SPIN. There are three possible verifying results for one property, which include: satisfaction, violation and out of memory.

3.2 System description

3.2.1 Getting the consistency verifying problems

The UML sequence diagrams have gained wide acceptance

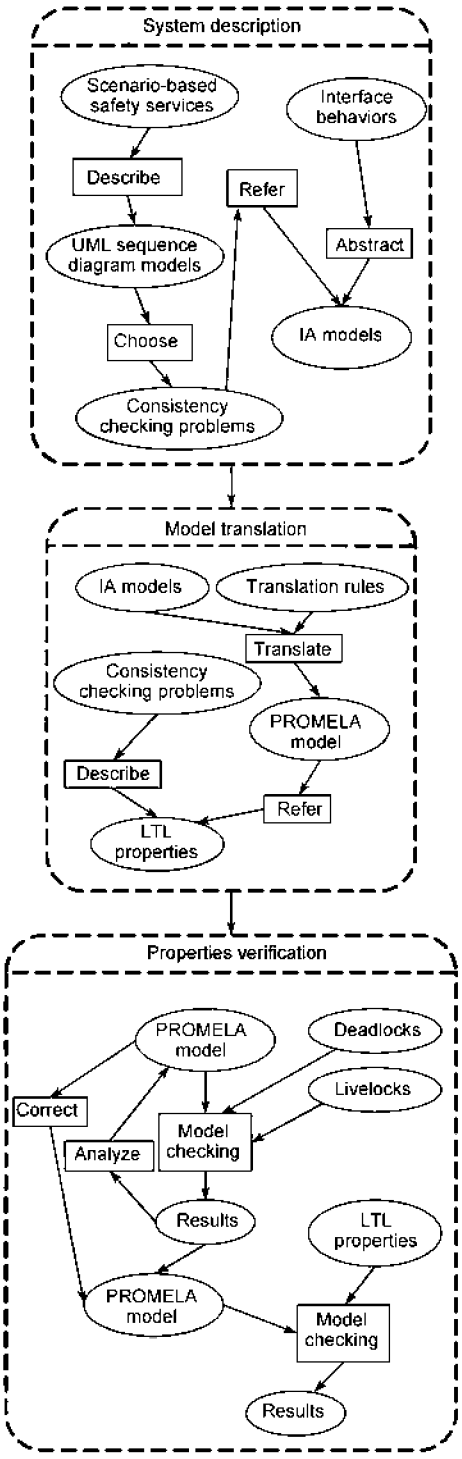


Figure 1 Waterfall model of safety protocols verification.

for scenario-based specifications of the component interactions [26]. They are used to show how entities interact in a system [27]. The entities can be, for example, subsystems, components, pieces of software, or users. In this study, the UML sequence diagrams are used to obtain the description of the scenario-based safety service interactions.

The problem of verifying the safety protocol specifica-

tions for the scenario-based safety service requirements is considered. Some consistency verifying problems can be selected from the UML sequence diagram models. Li et al. [28] developed the method to solve the following two verifying problems.

Existential consistency verifying problem, which means that a scenario described by a given UML sequence diagram must happen during a system run, or any forbidden scenario described by a given UML sequence diagram should never happen during a system run.

Forward mandatory consistency verifying problem, which means that if a reference scenario described by one given UML sequence diagram occurs during a system run, it must conform to a scenario described by the other given UML sequence diagram.

3.2.2 *Modeling the safety protocols using IA*

Model checking has to solve the problem of the explosion of the state space inevitably. The method proposed by Holzmann to solve this problem is to use design abstraction [29, 30], which is to assist the designer in the analysis of high-level abstractions without requiring the resolution of implementation-level detail. In our modeling method, the abstraction should be detailed enough to capture the essence of the specification and the IA model should be the smallest but sufficient model that allows designer to perform a verification of the interested properties.

In order to implement the modeling method, an $H=\{P_1,P_2,P_3,P_4,P_5\}$ is defined as a set of composable IA, which is used to describe the control-oriented abstraction [29] model of the safety protocols. P_1, P_5 are used to describe the safety service users of the two sides of communication respectively. They are upper layers of the safety layer. P_3 is used to model the transport layer below the safety layer, which is used to provide the transport services. P_2 and P_4 are defined as the two sides of the safety layers. $S=\{\text{shared}(P_i,P_j)\mid 1\leqslant i,j\leqslant 5, i\neq j\}$ is a set of all the shared actions in H. Two restrictions for H are defined as

Restriction 1.

$$\text{shared}(P_i,P_j)=\phi,(|i-j|>1). \tag{4}$$

Restriction 2.

$$\text{shared}(P_i,P_{i+1})\cap\text{shared}(P_{i+1},P_{i+2})=\phi,(1\leqslant i\leqslant 3). \tag{5}$$

Restriction 1 provides that there are no shared actions between two nonadjacent IA in H. Based on Restriction 1, Restriction 2 provides that every shared action in H must be shared only between two interface automata. It is not allowed for three or more interface automata to share the same actions.

As defined by Hu [31], a two-tuple $N=(H, S)$ which has been mentioned above is an IA network (IAN). Its illegal states set is defined as

$$\begin{aligned}\text{Illegal}(N)=\{&(v_1,v_2,\ldots,v_5)\in N\mid\exists(v_i,v_j)(i\neq j;1\leqslant i,j\leqslant 5),\\&\exists a\in\text{shared}(P_i,P_j),(a\in A_{P_i}^O(v_i)\wedge a\notin A_{P_j}^I(v_j)\\&\vee a\in A_{P_j}^O(v_j)\wedge a\notin A_{P_i}^I(v_i))\}.\end{aligned} \tag{6}$$

Since IA are not necessarily input-enabled, one of the automata may produce an output action that is not accepted by any other automata. Illegal(N) indicates the illegal states of N when this situation happens. In the real protocols, an illegal state can be reached and a deadlock may occur because of design defects in specifications. SPIN can be used to find these deadlocks by translating the IA model into the PROMELA model with the translation rules described below.

3.3 Model translation

3.3.1 *Translation rules from IA to the PROMELA model*

PROMELA allows for the dynamic creation of concurrent processes [32]. Communication via message channels can be defined to be synchronous (i.e., rendezvous), or asynchronous (i.e., buffered). The behavior of a process is defined by a proctype declaration. Each PROMELA model of a process can be described as a finite state automaton [29]. Mikk et al. [33] translated the UML statecharts into the PROMELA models and then realized the verification. But the translator did not support the full sublanguage of statecharts. Our translation rules support all the definitions of the elements of IA. Lilius et al. [34] proposed a method for translating the UML model into the PROMELA model, where each UML class was mapped to a PROMELA process. We map each interface automaton into a PROMELA process, and the translation rules are as follows:

Interface automaton P_i: Each interface automaton P_i should be mapped to a declaration of process, for example, proctype P_i(chan rcv).

The receiving channel of P_i: Each process should define a message channel CHchannelName, for example, chan CHchannelName=[0] of {mtype, …}. Particularly, the channels of an intermediate safety layer (P_2 and P_4) should be defined as an array. Array elements 0 and 1 are the channels for the SFM of the sponsor and the follower respectively.

State $v_i(0\leqslant i\leqslant|V_P|)$: Each state of an automaton should be mapped to a progress-state label, and the initial state labels should be put in the entry point of each process structure of proctype.

Input action $a\in A_P^I$: Each input action in an automaton should be mapped to a receiving statement, for example, rcv? variable1 (variable2, …).

Output action $a\in A_P^O$: Each output action in an automaton should be mapped to a sending statement, for example, channel name ! variable1 (variable2, …).

Internal action $a \in A_P^H$: Each internal action in an automaton should be mapped to a PROMELA statement except for the sending and receiving statements.

Step (v, a, v'): After an action v is executed, the state is transferred into v' by executing the statement "goto labelName", where labelName represents the progress-state label of v'.

In order to check the deadlocks caused by the reachable illegal states, the channel should be defined as a rendezvous port whose capacity is zero. A rendezvous port can only pass, but not store messages. Message interactions via such rendezvous ports are defined to be synchronous. If there is no matching receiving operation for it in other processes, the message sending statement in a process will be blocked. Meanwhile, the reachable illegal state which may lead to deadlocks could be found out by model checker SPIN via checking the invalid end-state as well.

3.3.2 Description of consistency verifying problems

Holzmann [29] promoted the frequently used LTL formulae to express the correctness properties. For example, [] p stand for that p is invariantly true. In this study, the method for describing the protocol specifications consistency verifying problems is as follows:

Existential consistency verifying problem. $\langle\rangle D$ is used to describe that scenario D is guaranteed to eventually happen at least once in a run; []!D is used to describe that scenario D never happens during a run.

Forward mandatory consistency verifying problem. $[](D_1 \rightarrow \langle\rangle D_2)$ is used to describe that if scenario D_1 happens, D_2 is guaranteed to eventually happen at least once in a run.

3.4 Properties verification

Two important model correctness criterions are the absence of deadlocks and livelocks. They should be checked before verifying the LTL described properties. Some IA may very well linger in a known waiting state, or they may sit patiently in a loop ready to spring back to action when new input arrives. To make it clear to the verifier that these alternate end states are also valid, some special labels can be defined which are called end-state labels. The end-state label defines that it is not an error, if the process waits at the label [29] in the end of an execution sequence. If the translation rules defined above are used, the reachable illegal states in IA model which may lead to deadlocks will be found by SPIN via checking the invalid end-state of the model. The livelocks in PROMELA can be detected using non-progress loops [29]. A livelock is detected, when the state space contains a cycle that leads to no way outside the cycle. SPIN can verify that every permitted infinite execution cycle that passes through at least one of the progress labels in that model. If cycles can be found not to have this property, the verifier can declare the existence of a non-progress loop, corresponding to possible starvation.

4 Case study: verification of a safety protocol

4.1 A safety communication protocol in TCS

The ERTMS consortium has defined a set of specifications for the European high-speed railways in which traditional signaling has been replaced by radio signaling via global system mobile-railway (GSM-R). In the ERTMS level 2 radio communication system, an on-board system transmits the train position report to a radio block center (RBC). Train separation is managed by RBC that transmits the movement authority (MA) with static speed profile (SSP) to the trains via radio regarding the position, conditions of railway line, etc. The safety of passengers depends on the reliability of the communication system. The safety protocol thus plays a major role in the implementation of the TCS. Since GSM-R is not able to guarantee an acceptable safety level for the critical data transmission, it is necessary to add safety-related transmission functions upon the non-trusted channel according to IEC 62280-2 [35] and the safety protocol of EURORADIO [36] .

The EURORADIO safety protocol is a layered protocol [2] using service primitives to interact between different layers. The safety service structure is shown in Figure 2. The service interface between the safety functional module (SFM) and the safety service (SaS) user gives the data flows to/from the SFM, which provides safety services. The SaS user exchanges data with the SaS provider to get the safety services which provide safety connection set-up, and safety data transfer during the connection lifetime. The SFM reports the errors that occur in the safety layer and transfers the error indications from the lower layers. A safety entity of the SFM communicates with its users through one or more safety service access points (SaSAP) using the safety service primitives. For example, the safety service primitive Sa-CONN.req represents a safety connection set-up request, where the symbol "Sa" represents safety and the symbol "CONN.req" represents connection request. The peer safety entities of the SFM support the safety connection exchanges by means of safety protocol data units (SaPDU), using the services of the transport layer via one transport connection (TC) through one transport service access point (TSAP). For example, AU1, AU2 and AR represent the SaPDUs of first authentication message, second authentication message and authentication response respectively. Communication functional module (CFM) is used to provide transport service (TS) using transport service primitive, and the safety entity plays the role of a TS user. For example, the transport service primitive T-CONN.req means a transport connection request to CFM, where the symbol "T" represents transport and "CONN.req" represents a TC request. The symbols "DISC", "CONN", "DATA" and "HP-DATA"

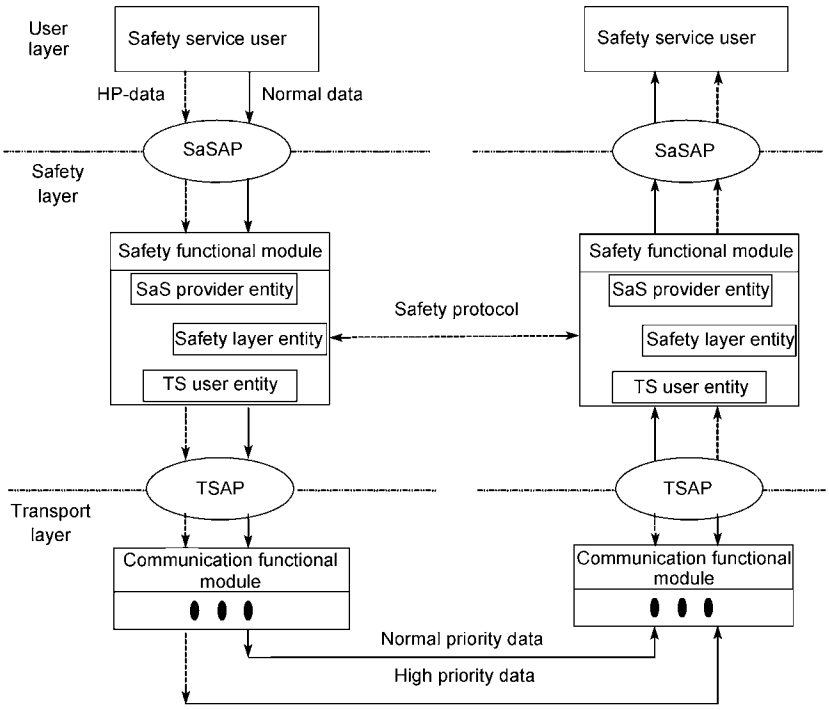


Figure 2 Safety service structure of ETCS communication protocol.

represent connection disconnect, connect set-up, data transfer and high priority (HP) data transfer respectively, while the symbols "req", "ind", "resp" and "conf" represent request, indication, response and confirmation respectively. These symbols are used to make up all the primitives, which are consistent with the acronyms and abbreviations described in the EURORADIO functional interface specification [36].

4.2 A UML sequence diagram model of the safety protocol in TCS

Figure 3 shows a UML sequence diagram representing the simple interactions between six entities of two communication parties. This sequence diagram shows a scenario where one communication side establishes a safety connection with another side. The process of establishing a safety connection is initiated when the SaS user requests a connection to the SFM using the service primitive Sa-CONN.req. Then the safety layer requests transport connection establishment using the service primitive T-CONN.req. This service primitive includes the first message of the peer entity authentication procedure (AU1 SaPDU) as user-data. The SFM will send the AU1 SaPDU to the SFM of the other side using the transport connection request service primitive

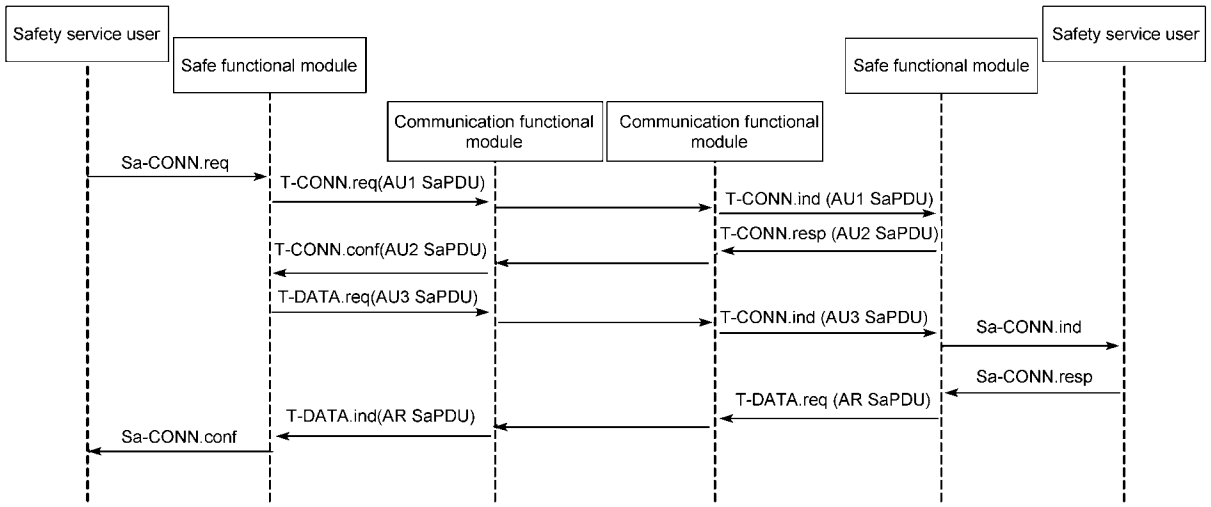


Figure 3 UML sequence diagram for safe connection set-up of normal execution.

through the CFMs. If it is accepted, the safety entity responds to the TC establishment request using the service primitive T-CONN.resp. It includes the second message of the peer entity authentication protocol (AU2 SaPDU). Once the messages are received, the calling CFM informs the safety layer of the successful establishment of the transport connection using the service primitive T-CONN.conf. The AU2 SaPDU is forwarded to the safety layer as user-data within this service primitive. SFM then generates the AU3 SaPDU that contains the third message of the authentication protocol. It uses the T-DATA.req service primitive to forward this message to the transport layer. Once the messages are received, CFM uses the service primitive T-DATA.ind to forward the AU3 SaPDU to the safety layer. In the case of a successful AU3 SaPDU evaluation, the safety entity forwards the service primitive Sa-CONN.ind to the safety user. If the safety user accepts the safety connection establishment request, it responds with the primitive Sa-CONN.resp. The safety entity on the called side sends the authentication response message including the authentication data (AR SaPDU) to its peer safety entity.

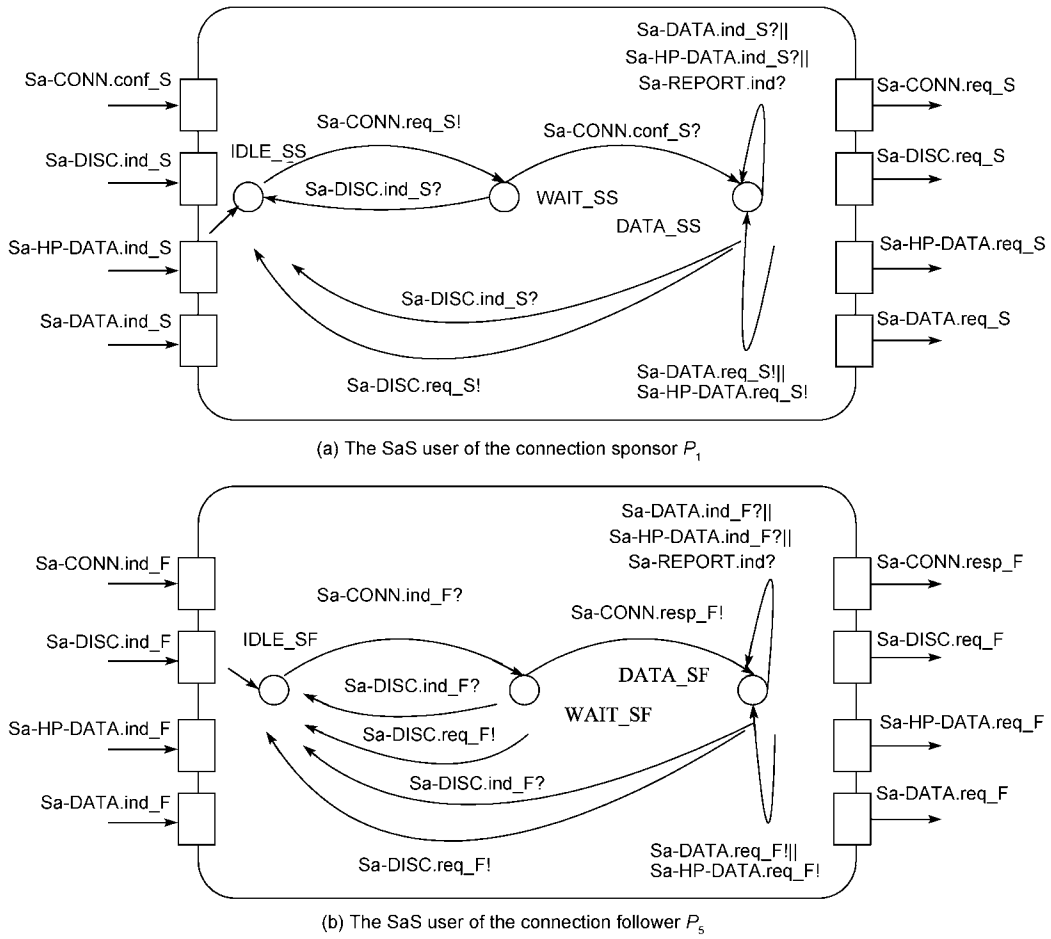
After a successful evaluation of this SaPDU, the safety entity informs the SaS user that a safety connection is now successfully established, using the service primitive Sa-CONN.conf. A maximum connection establishment delay timer T_{estab} is used by the SFM of the calling side for detecting unacceptable delay during the connection establishment. The timer T_{estab} is started after the receipt of the Sa-CONN.req and is stopped before the generation of the Sa-CONN.conf. In the case of timeout, a Sa-DISC.ind is generated including a proper reason.

4.3 An IA model of the EURORADIO safety protocol

Following the method presented above, $H=\{P_1, P_2, P_3, P_4, P_5\}$ is used to describe the abstracted model of the safety protocol. P_1, P_2, P_3, P_4, P_5 represent the SaS user of the connection sponsor, the SFM of the connection sponsor, the transport layer, the SFM of the connection follower and the SaS user of the connection sponsor respectively. P_1 and P_5 are illustrated in Figure 4. The inner structures of P_2 and P_4 are shown in Figure 5. P_3 is shown in Figure 6.

The descriptions of P_1, P_5, P_2, P_4 and P_3, are shown as follows:

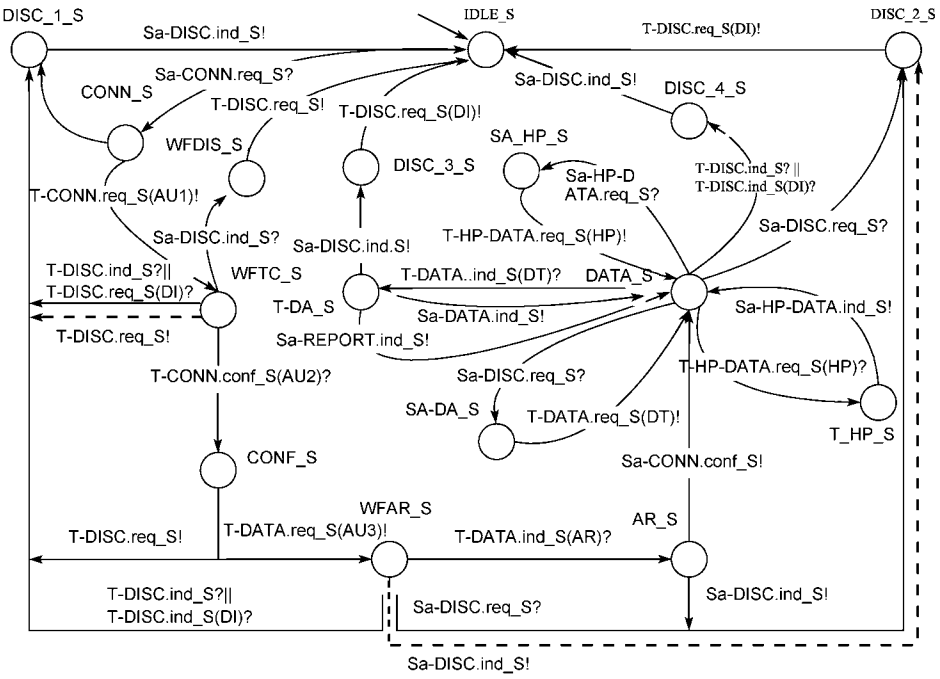
SaS user of the sponsor P_1: P_1 sends the safety connection set-up request to P_2 using the safety service primitive



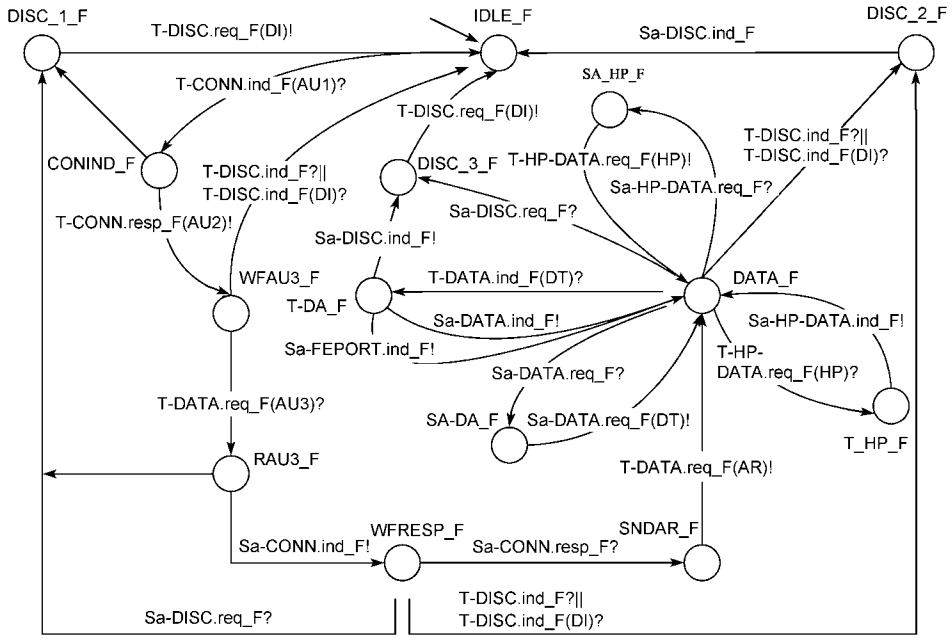
(a) The SaS user of the connection sponsor P_1

(b) The SaS user of the connection follower P_5

Figure 4 Interface automaton models of P_1 and P_5.



(a) The inside structure of the interface automaton for the SFM of the connection sponsor P_2



(a) The inside structure of the interface automaton for the SFM of the connection follower P_4

Figure 5 IA models of P_2 and P_4.

Sa-CONN.req. After P_1 receives the safety connection set-up confirmation primitive Sa-CONN.conf, it can transmit either normal data or HP data depending on the type of service primitives. Sa-DATA.req_S! || Sa-HP-DATA.req_S! means a safety connection sponsor sends a Sa-DATA.req or Sa-HP-DATA.req to P_2.

SaS user of the follower P_5: After P_5 receives the safety connection set-up indication primitive Sa-CONN.ind, it sends the safety connection response primitive to P_4 and then P_5 reaches the data transmitting state. In this state, either normal data or HP data can be transmitted by P_5.

SFM of the connection sponsor P_2: During the state IDLE_S, P_2 waits for the safety connection set-up request primitive. The process of establishing a safety connection is initiated when P_2 receives a safety connection set-up request from P_1 via the primitive Sa-CONN.req. If the Sa- CONN.req

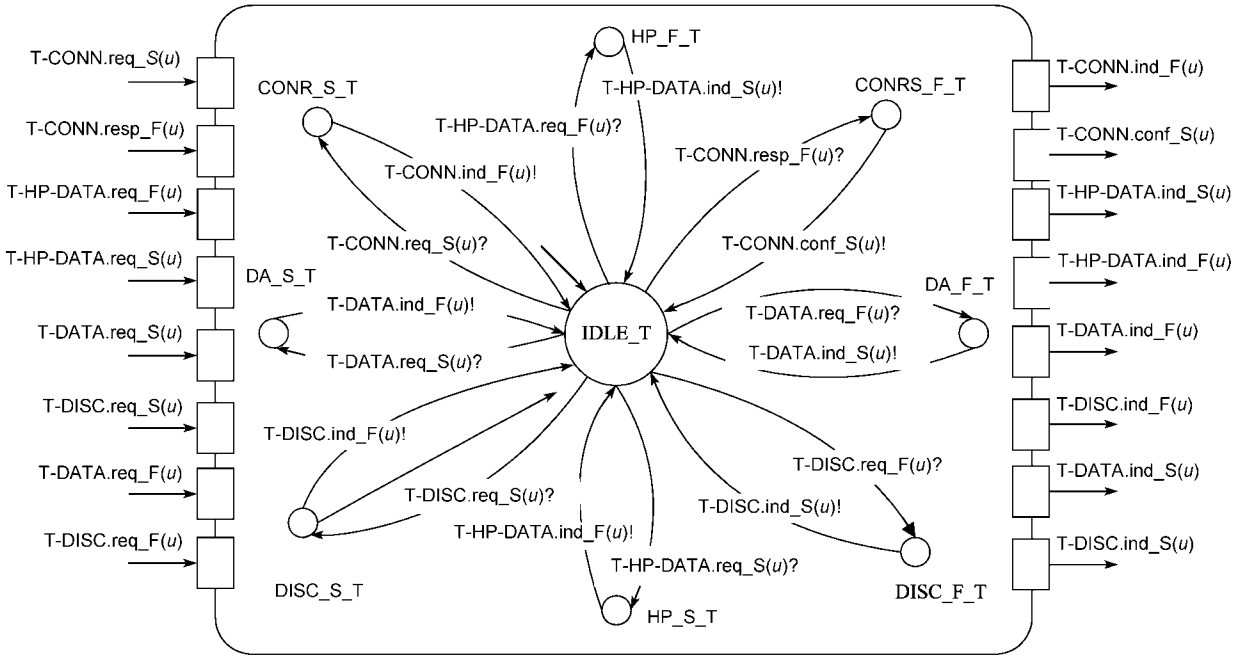


Figure 6 IA model of TS layer P_3.

is not reachable, P_2 shall send the safety connection disconnection primitive Sa-DISC.ind to P_1 and return to the state IDLE_S, otherwise P_2 can send the AU1 SaPDU to P_4 via the transport connection request primitive T-CONN.req. If P_2 receives the right AU2 SaPDU from P_4, it will send the third authentication AU3 SaPDU to P_4, and then P_2 starts to wait for the authentication response (AR SaPDU) in the state WFAR_S. In the states WFTC_S and WFAR_S, if the safety connection establishment time exceeds the upper bound, P_2 needs to send the safety disconnection indication primitive Sa-DISC.ind to P_1 and the transport disconnection request primitive T-DISC.req to P_3, and then returns to the state IDLE_S. The two dashed arcs in P_2 are used to model the response events when T_{estab} is up. In the state WFAR_S, if P_2 receives the right AR SaPDU, it will send the safety connection confirmation primitive Sa-CONN.conf and reach the state DATA_S, then the safety connection is established successfully. In the state DATA_S, P_2 can communicate with P_4 using data SaPDU DT and high priority data SaPDU HP. If P_2 receives the safety disconnection request primitive Sa-DISC.req from P_1 or the transport disconnection indication primitive T-DISC.ind from P_3, it will return to the state IDLE_S.

SFM of the connection follower P_4: At the very beginning, P_4 stays in the state IDLE_F, waiting for the transport connection indication primitive T-CONN.ind with AU1 SaPDU. If P_4 receives the unacceptable primitive T-CONN.ind or SaPDU AU1, it will send the transport disconnection request primitive T-DISC.req with DI SaPDU to P_3 and return to the state IDLE_F. If P_4 receives the acceptable primitive T-CONN.ind and SaPDU AU1, it will send the transport connection response primitive T-CONN.resp with AU2 SaPDU to P_3 and reach the state WFAU3_F. After P_4 receives the acceptable transport data request primitive T-DATA.req with AU3 SaPDU and sends the safety connection indication primitive Sa-CONN.ind to P_5, it will reach the state WFRESP_F and wait for the safety connection response primitive Sa-CONN.resp. After P_4 sends the AR SaPDU to P_2 through P_3, it will reach the state DATA_F. P_4 can communicate with P_2 using the DT SaPDU or the HP SaPDU depending on the data priority.

TS Layer P_3: P_3 is used to receive transport service primitives from one side of the safety layer and sends the corresponding primitives to the other side. For example, after P_3 receives the transport connection set-up request primitive T-CONN.req from P_2, it sends the transport connection set-up indication primitive to P_4. There are two connections from the state DISC_S_T to the state IDLE_T, in which the one without any actions models the data lost in the TS layer. T-DISC.ind_F(u) represents the transport primitive T-DISC.ind_F which contains a SaPDU u.

The purpose of the protocol verification is to find the design defects and verify whether some properties are satisfied. It is better to describe some abnormal situations in the interface automaton of the safety layer. For example, if a message error occurs during transmission, a CRC error will be checked when the receiver gets the message. This kind of abnormal situation should be described in IA. The output action T-DISC.req_S transited from CONF_S to IDLE_S stands for the situation that an error occurs in the message transmitting process. When the SFM of the connection sponsor receives the unacceptable AU2 SaPDU, the SFM sends the disconnect requirement to the transport layer.

4.4 Verification of the safety communication protocols

One aim of verification is to verify if the protocol is provided with some properties, and gives the route of the counterexamples. In order to illustrate the effect of the presented method on verifying deadlocks, livelocks and several mandatory consistency properties, some properties are presented as an example.

Property 1 (Deadlock). The deadlocks can be detected via checking the invalid end-states of the model.

Property 2 (Livelocks). The livelocks can be detected using non-progress loops.

Property 3 (Forward mandatory consistency). The SaS user of the sponsor will be able to receive the connection confirmation after it sends the safety connection request.

LTL:

$[](SaUserS\_SaConnReq \rightarrow \langle\rangle\, SaUserS\_SaConnConf)$

SaUserS_SaConnReq means that the SaS user of the sponsor is in the connection requesting sub-state;

SaUserS_SaConnConf means that the SaS user of the sponsor is in the receiving connection confirmation sub-state.

Property 4 (Existential consistency). The SaS user will never receive the disconnection indication from the under layer during a PROMELA run.

LTL: $[]!\ (SaUserS\_SaDiscInd)$

SaUserS_SaDiscInd means that the SaS user of the sponsor is in the receiving disconnection indication sub-state.

Property 5 (Forward mandatory consistency). After the SaS user of the sponsor sends the service primitive SaHpDataReq, the SFM of the sponsor will be able to send the primitive THpDataReq.

LTL:

$[](SaUserS\_SaHpDataReq \rightarrow \langle\rangle\ SaLA\_THpDataReq)$

SaUserS_SaHpDataReq means that the SaS user of the sponsor is requested to send HP data;

SaLA_THpDataReq means that the SFM of the sponsor is requested to send HP data.

Property 6. The SFM of the sponsor is always in the state IDLE_SS until it receives the service primitive SaDiscReq; the SFM of the follower is always in the state IDLE_SF until it receives the service primitive TConnInd.

LTL:

$SaLA\_IDLE \bigcup SaLayA\_SaConnReq$

$SaLB\_IDLE \bigcup SaLayB\_TConnInd$

SaLA_IDLE means that the SFM of the sponsor is in the idle state;

SaLayA_SaConnReq means that the SFM of the sponsor is in the connection requesting sub-state;

SaLB_IDLE means that the SFM of the follower is in the idle state;

SaLayB_TConnInd means that the SaS user of the follower is in the receiving connection indication sub-state.

SPIN's typical working mode is to start with the specifications of a high level model of a concurrent system, or distributed algorithm, typically using SPIN's graphical front-end XSPIN [29]. All the experiments are done on a 2.4 GHz Pentium 4 processor, with 512 MB of main memory. At the very beginning, the IA model of H without modeling the timer of T_{estab} is verified. In this case, the two dashed arcs in P_2 do not exist.

Using the XSPIN's model checking function, a deadlock is checked out. Even though end-state labels are added to the right end states, an invalid end state can be found. The trace of H, which leads to the deadlock, is analyzed and acquired using the SPIN's counterexample. As shown in Figure 7, the safety connection establishing process is started in the 1st step. The abnormal situation starts in the 6th step, where H={P_1, P_2, P_3, P_4, P_5} is in the state (WAIT_SS, CONF_F, IDLE_T, WFAU3_F, IDLE_SF). Because of the unacceptable T-CONN.ind or AU1 SaPDU, P_2 sends the transport disconnection primitive of T-DISC.req to P_3. After P_3 receives the primitive, H reaches the state (WAIT_SS, DISC_1_S, DISC_S_T, WFAU3_F, IDLE_SF). When the data is lost in the TS layer, P_3 will return to the state IDLE_T without sending the transport disconnection indication to P_4, and H will reach the state (WAIT_SS, DISC_1_S, IDLE_T, WFAU3_F, IDLE_SF). After P_1 receives the safety disconnection indication, it returns to the initial state and attempts to establish a new safety connection again. In the 11th steps, H reaches to the state (WAIT_SS, WFTC_S, CONR_S_T, WFAU3_F, IDLE_SF). In this state, P_3 attempts to indicate the transport connection by sending T-CONN.ind, but P_4 does not accept it in state WFAU3_F. According to Definition 3, it is a reachable illegal state. There is no any other executable activity, and these processes are blocked. There are two methods that can make the system continue the execution. One is to add a safety disconnection function to the state WAIT_SS of P_1, the other is to add the function of T_{estab} responding to P_2 (the two dashed arcs).

Based on these results, a conclusion can be drawn that T_{estab} also has the function of avoiding deadlocks in the safety communication protocol. So it is a mistake to drop the response events of T_{estab} in the abstraction model. Another mistake in H is a lack of the safety disconnection function in the state WAIT_SS of P_1. After the arc from the state WAIT_SS to the state IDLE_SS is added to P_1 and the two dashed arcs are added to P_2, the mistakes in H are corrected and a new PROMELA model is ready for the promoted H. The properties of the promoted H model checking results are given below.

Properties 1 and 2 are satisfied. The end-state and the progress labels are added to the right end states and cycling states. The satisfaction to the properties 1 and 2 is crucial for correctly expressing the LTL formulae which are used to verify the following properties.

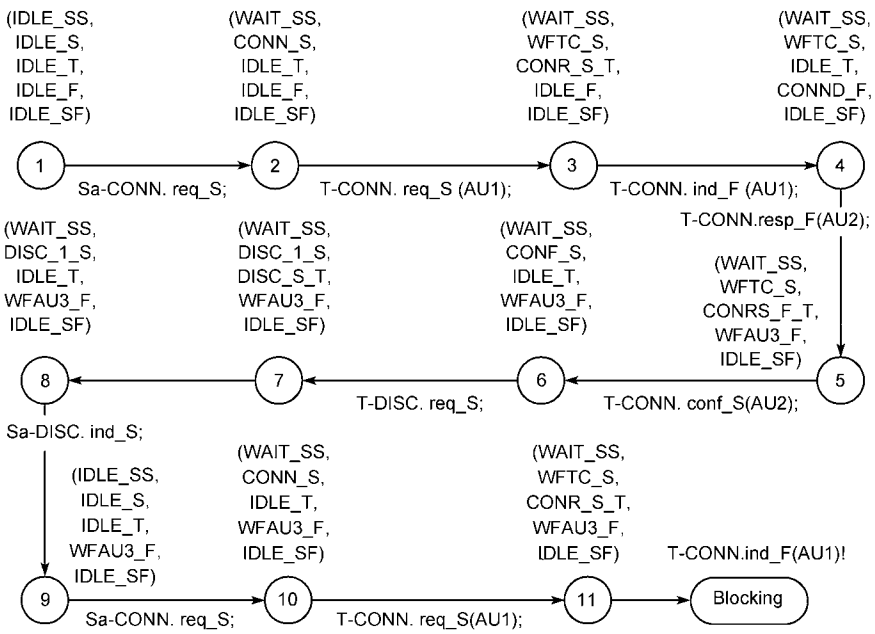


Figure 7 A trace leading to the deadlock.

Property 3 is not satisfied. Through a given counterexample, we find that the safety connection set-up process fails after the SaS user of the sponsor receives the connection request service primitive with an error format.

Property 4 is not satisfied. Both the SaS user of the sponsor and follower are guaranteed to eventually receive the disconnection indication from an underlayer at least once during a PROMELA run.

Properties 5 and 6 are satisfied.

According to these model checking results, two crucial features of the SFM layer are revealed. Firstly, it may never receive the connection confirmation after the SaS user of the sponsor sends the safety connection request. It is guarantees that the safety protocol does not satisfy the forward mandatory consistency of Property 3. This is because there may be deletion or modification of the messages sent from a RBC to a train. The failure of the safety connection set-up process must be reported to the SaS user using disconnection indication primitive. Secondly, the safety layer cannot guarantee that the safe connection is kept all the time. It is because that the safety protocol is built upon the non-trusted "open" channel. For example, the communication connection between a train and a RBC may be disconnected because of the breakdown of GSM-R. In the situation of abnormal communication disconnection persisting beyond a reasonable time, the SaS user needs to make safety defences such as an emergency brake.

5 Application and performance

The formal verification method presented above is used during the development process of a safety communication protocol in the communication based train control (CBTC) systems [7]. During the debug process of a first version (version 1) implement of the safety communication protocol in the CBTC system, some hidden problems are very hard to be found from the massive amounts of code. For example, the safety connection cannot be established if it is disconnected after a long time. Some of the reasons are that the designed specification does not satisfy some safety application requirements [37]. To solve this problem and reduce the code debugging time, the formal verification method presented above is used during the development of the second version (version 2) of the safety communication protocol in the CBTC system. The specification of the safety communication protocol is described using IA and verified by the model checker SPIN. According to the formal description of IA, the safety protocol is refined and implemented using the C language over Vxworks real-time operating system above the UDP layer. The experiment results show that the design cycle is shortened, and the development and debugging efficiency are enhanced. The real-time performance of the advanced protocol is obtained using network simulation tool OPNET [7, 38]. The simulation results show that the real-time performance satisfies the safety requirements.

To analyze the performance of two protocols in terms of safety, the test is performed under the same conditions. The direct impact of the potential safety hazards is data losses of channel, which result in a disconnection of safety connection. Therefore, in this paper, we present safety connection disconnection rate with different data losses according to the test conditions presented in Table 1.

Table 1 Test conditions of the safety communication protocols

Test conditions	Values
Communication cycle	0.2 (s)
Times of safety connection	5000
Application message length	100 (byte)
Transport layer protocol	UDP
Standard of Ethernet	IEEE 802.3
Transmutation rate of Ethernet	100 (Mbit s^{-1})

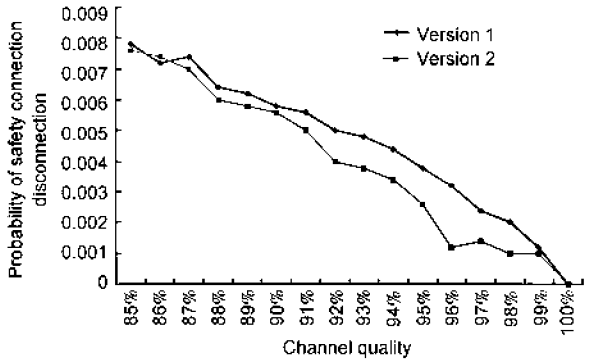


Figure 8 Comparison of probability of safety connection disconnection.

Figure 8 shows the characteristics of abnormal disconnections in these two protocols when the channel quality of transport layer is from 85% to 100%. As observed from the figure, the protocol of version 2 always gives a better safety connection, which further proves the effectiveness of our method.

6 Conclusion

In this study, a formal description method is developed for the TCS safety communication protocols using IA. The safety service interface behaviors and scenario-based interactions of the safety protocols are described using IA and UML sequence diagram. How IA can be translated into PROMELA language for the verification of the safety protocols has also been presented. In our method, SPIN is used for the verification of deadlocks, livelocks and some mandatory consistency properties. However, the method is not limited to these properties. Since IA are translated into PROMELA, the full range of the SPIN verification system is variable. Once validated and checked, this specification can be used as a base for a design and implementation.

The feasibility of the method is demonstrated by applying it to an example. The preliminary results show both potential efficiency and practical utility of the method. The causes of the deadlocks in the specification of the safety protocols in TCS are found out using the counterexample of SPIN. This formal verification method has been used to develop the CBTC safety communication protocol. This protocol has already been applied to Beijing subway Yizhuang line and Changping line. It is also expected that the development of a new safety communication protocol will be significantly accelerated with the help of the proposed method, and the safety level will be remarkably enhanced as well. As future work, we plan to develop an automatic translator from IA to PROMELA model based on the proposed translation rules.

This work was supported by the New Century Excellent Researcher Award Program from Ministry of Education of China (Grant No. NCET-07-0059), the Fundamental Research Funds for the Central Universities (Grant No. 2011YJS006), the National High Technology Research and Development Program of China ("863" Program) (Grant No. 2011AA010104), and the State Key Laboratory of Rail Traffic Control and Safety Research Project (Grant Nos. RCS2008ZZ001, RCS2008ZZ005).

1 Heimdahl M P E. Safety and software intensive systems: challenges old and new. In: Conformance of Future of Software Engineering, 2007. Washington: IEEE Computer Society, 2007. 137–152
2 Esposito R, Sanseviero A, Lazzaro A, et al. Formal verification of ERTMS euroradio safety critical protocol. In: Proceedings of FORMS 2003. Budapest: IEEE Computer Society, 2003. 21–29
3 Diao Y F, Wang B D. Risk analysis of flood control operation mode with forecast information based on a combination of risk sources. Sci China Tech Sci, 2010, 53(7): 1949–1956
4 Chu Y Y, Zhang H, Shen S F, et al. Development of a model to generate a risk map in a building fire. Sci China Tech Sci, 2010, 53(10): 2739–2747
5 Xu T H, Tang T, Gao C H, et al. Dependability analysis of the data communication system in train control system. Sci China Tech Sci, 2009, 52(9): 2605–2618
6 Gronbaek J, Madsen T K, Schwefel H P. Safe wireless communication solution for driver machine interface for train control systems. In: Proceedings of International Conference on Systems (ICONS 2008). Cancun: IEEE Computer Society, 2008. 208–213
7 Zhang Y, Tang T, Yan F. Study on model for analysis of CBTC data communication system (DCS) and its application (in Chinese). J China Railway Soc, 2011, 33(5): 60–65
8 Sinha P, Ren D Q. Formal verification of dependable distributed protocols. Inf Software Technol, 2003, 45(12): 873–888
9 Clarke E M, Wing J M. Formal methods: state of the art and future directions. ACM Computing Surveys, 1996, 28(4): 626–643
10 Lee J H, Hwang J G, Park G T. Performance evaluation and verification of communication protocol for railway signaling systems. Computer Standards & Interfaces, 2005, 27(3): 207–219
11 Lee J D, Jung J I, Lee J H, et al. Verification and conformance test generation of communication protocol for railway signaling systems. Computer Standards & Interfaces, 2007, 29(2): 143–151
12 Lee J H, Hwang J G, Shin D, et al. Development of verification and conformance testing tools for a railway signaling communication protocol. Computer Standards & Interfaces, 2009, 31(2): 362–371
13 Katsaros P. A roadmap to electronic payment transaction guarantees and a Colored Petri Net model checking approach. Inf Software Technol, 2009, 51(2): 235–257
14 Sinha P, Suri N. Modular composition of redundancy management protocols in distributed systems: an outlook on simplifying protocol level formal specification and verification. In: 21st International Conference on Distributed Computing Systems. Phoenix: IEEE Computer Society, 2001. 255–263
15 Sinha P, Suri N. On simplifying modular specification and verification of distributed protocols. In: Sixth IEEE International Symposium on High Assurance Systems Engineering. Boca Raton, Florida: IEEE Computer Society, 2001. 173–181
16 Ouzzif M, Erradi M, Mountassir H. Description of a teleconferencing floor control protocol and its implementation. Eng Appl Artif Intel,

2008, 21(3): 430–441
17 Schäfer T, Knapp A, Merz S. Model checking UML state machines and collaborations. Elec Notes Theor Comp Sci, 2001, 55(3): 357–369
18 Inverardi P, Muccini H, Pelliccione P. Automated check of architectural models consistency using SPIN. In: Proceeding of the 16th IEEE International Conference on Automated Software Engineering (ASE 2001). Los Alamitos: IEEE Computer Society, 2001. 346–349
19 Alfaro L, Henzinger T A. Interface automata. In: 8th Eiropean Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9). Vienna: ACM Press, 2001. 109–120
20 Alfaro L D, Henzinger T A. Interface theories for component-based design. In: Proceedings of the First International Workshop on Embedded Software. Tahoe City, CA: Springer, 2001. 148–165
21 Jin Y, Esser R, Lakos C, et al. Modular analysis of dataflow process networks. In: Joint European Conferences on Theory and Practice of Software. Warsaw: Springer, 2003. 184–199
22 Chakrabarti A, De Alfaro L, Henzinger T, et al. Interface compatibility checking for software modules. In: Proceedings of the 14th International Conference on Computer-Aided Verification. Copenhagen: Springer, 2002. 428–441
23 Chakrabarti A, Alfaro L D, Henzinger T A, et al. Synchronous and bidirectional component interfaces. In: Proceedings of the 14th International Conference on Computer Aided Verification. Copenhagen: Springer, 2002. 414–427
24 Lee E A, Xiong Y, Behavioral types for component-based design. Technical Report No. UCB/ERL M02/29, Berkeley, USA, 2002
25 Kapus T. Using mobile TLA as a logic for dynamic I/O automata. IEICE Trans Inf Syst, 2009, 92(8): 1515–1522
26 Refsdal A, Stølen K. Extending UML sequence diagrams to model trust-dependent behavior with the aim to support risk analysis. Sci Comp Progr, 2008, 74(1-2): 34–42
27 Medvidovic N, Rosenblum D S, Redmiles D F, et al. Modeling software architectures in the Unified Modeling Language. ACM Trans Software Eng Methodol, 2002, 11(1): 2–57
28 Li X D, Hu J, Bu L, et al. Consistency checking of concurrent models for scenario-based specifications. In: 12th International SDL Forum, SDL 2005: Model Driven, Grimstad. Berlin: Springer 2005. 1171–1180
29 Holzmann G J. The model checker SPIN. IEEE Trans Software Eng, 1997, 23(5): 279–295
30 Wang Y, Wei J, Wang Z Y. Model checking distributed control systems based on software architecture (in Chinese). J Software, 2004, 15(6): 823–833
31 Hu J, Yu X F, Zhang Y, et al. Checking component-based designs for scenario-based specifications (in Chinese). Chin J Comp, 2006, 29(4): 513–525
32 Bharadwaj R, Heitmeyer C L. Model checking complete requirements specifications using abstraction. Autom Software Eng, 1999, 6(1): 37–68
33 Mikk E, Lakhnech Y, Siegel M, et al. Implementing statecharts in PROMELA/SPIN. In: Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques. Florida: IEEE Computer Society, 1998. 90–101
34 Lilius J, Paltor I P. VUML: a tool for verifying UML models. In: 14th IEEE International Conference on Automated Software Engineering (ASE'99). Florida: IEEE Computer Society, 1999. 255–258
35 IEC, IEC 62280-2, Railway applications-communication, signaling and processing systems-part 2: safety-related communication in open transmission systems. New York: IEC, 2001
36 ERTMS/ETCS UNISIG Subset-037: Euroradio FIS. http://www.era.europa.eu/Document-Register/Documents/Subset-037%20v230.pdf. 2005
37 Zhang Y, Zhao X Q, Zheng W, et al. System safety property-oriented test sequences generating method based on model checking. WIT Trans Built Environ, 2010, 144(1): 747–759
38 Zhang Y, Tang T, Ma L C, et al. Modeling and simulation of the security communication protocol based on the switched Ethernet (in Chinese). J China Railway Soc, 2010, 32(3): 43–48